

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 06-222928

(43)Date of publication of application : 12.08.1994

(51)Int.Cl.

G06F 9/45

(21)Application number : 05-011369

(71)Applicant : HITACHI LTD
HITACHI MICOM SYST:KK

(22)Date of filing : 27.01.1993

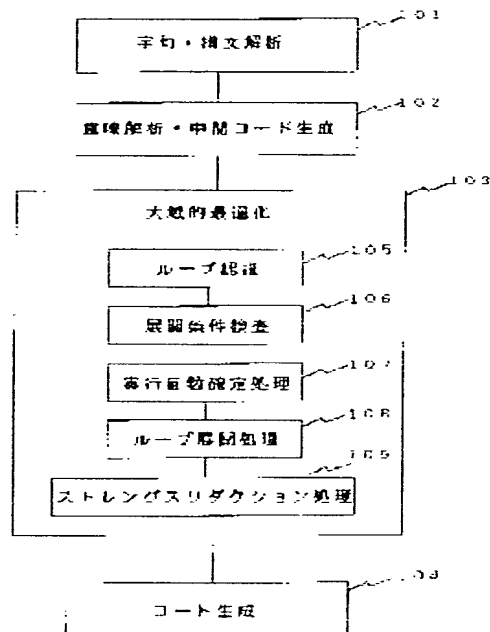
(72)Inventor : TOYAMA KEISUKE
HASHIMOTO AKIRA

(54) LOOP OPTIMIZING METHOD

(57)Abstract:

PURPOSE: To provide a system for improving execution performance by improving the possibility of loop extension and further for executing valid strength reduction even at an RISC computer, for which the use of an index mode is limited, as well, by utilizing that extension concerning the optimizing method for a program.

CONSTITUTION: Concerning a loop, an executing time fixing processing circuit 107 confirms that the number of times of repetition is not changed inside the loop. A loop extension processing part 108 decides the number of times of execution at the time of execution and generates a correspondent extended part together with a code to be branched. A strength reduction processing part 109 further optimizes the extended loop. Thus, even the loop, which can not decide the number of times of repetition at the time of compile, can be extended as well and the optimizing effect of loop extension can be provided. Further, strength reduction can be executed to the main body of the extended loop as well and even in the case of a limited addressing mode observed in a lot of RISC computers, arithmetic is decreased.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁 (J P)

(12) 公 開 特 許 公 報 (A)

(11)特許出願公開番号

特開平6-222928

(43)公開日 平成6年(1994)8月12日

(51)Int.Cl.⁵

G 0 6 F 9/45

識別記号

庁内整理番号

F I

技術表示箇所

9292-5B

G 0 6 F 9/ 44

3 2 2 G

審査請求 未請求 請求項の数 7 O L (全 10 頁)

(21)出願番号

特願平5-11369

(71)出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(22)出願日

平成5年(1993)1月27日

(71)出願人 000233169

株式会社日立マイコンシステム

東京都小平市上水本町5丁目22番1号

(72)発明者 十山 圭介

神奈川県川崎市麻生区王禅寺1099番地 株

式会社日立製作所システム開発研究所内

(72)発明者 橋本 明

東京都小平市上水本町5丁目22番1号 株

式会社日立マイコンシステム内

(74)代理人 弁理士 小川 勝男

(54)【発明の名称】 ループ最適化方法

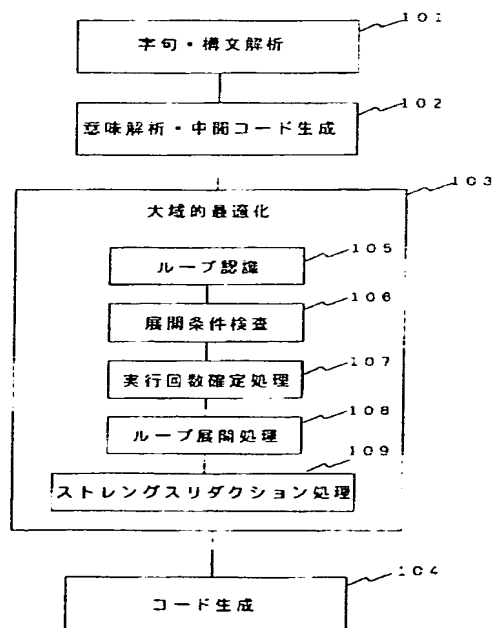
(57)【要約】

【目的】 本発明の目的は、プログラムの最適化手法において、ループ展開の可能性を高めて実行性能を向上させ、さらにその展開を利用して、インテックスモードの使用に制限のあるRISC計算機においても有効なストレングスリダクションを実施する方式を提供することにある。

【構成】 ループに対し、実行回数確定処理部(106)で繰返し回数がループ内で変更されないことを確認する。ループ展開処理部(107)では実行回数を実行時に判定し分岐するコードとともに対応する展開部を生成する。ストレングスリダクション処理部(108)で展開後のループをさらに最適化する。

【効果】 コンパイル時に繰返し回数の決定できないループも展開でき、ループ展開の最適化効果を得ることかてきる。また展開されたループ本体に対してストレングスリダクションを実施することも可能になり、多くのRISC計算機にみられる制約されたアトレンシクモードである場合にも演算を減少させる。

図1



【特許請求の範囲】

【請求項1】 計算機言語で記述されたプログラムのソースコードから実行用のオブジェクトコードを生成するコンパイラにおいて、繰返し構造を表現しているループのプログラムソースをオブジェクトコードに変換する方法であって、

コンパイル時には該ループの繰返しの実行回数が不明であるが、実行時に該繰返し回数が、そのループの入口で確定するものに対して、該ループの入口において繰返し回数を実行時に判定し、それに応じて繰返すべき回数だけ、もしくはそれを複数の数の和として表し、それぞれ該複数の数だけループ本体の複製を作成することによって展開を実施することを特徴とするループ最適化方法。

【請求項2】 請求項1記載のループ最適化方法において、該ループの入口における実行時の繰返し回数の判定を、その回数の偶数・奇数によって行い、それに応じて奇数回繰返しの場合と偶数回繰返しの場合とに分けて、繰返すべきループ本体の複製を作成し、展開することを特徴とするループ最適化方法。

【請求項3】 計算機言語で記述されたプログラムにおいて繰返し構造を表現しているループに関して、その実行速度を向上させるためのプログラム変換手法を、請求項1記載の方法において展開された後のループの繰返し本体に適用することを特徴とするループ最適化方法。

【請求項4】 計算機言語で記述されたプログラムにおいて繰返し構造を表現しているループに関して、その実行速度を向上させるプログラム変換手法の一つであるストレンクスリタクション方式を、一つのループに対して複数回適用することを特徴とするループ最適化方法。

【請求項5】 計算機言語で記述されたプログラムにおいて繰返し構造を表現しているループに関して、その実行速度を向上させるプログラム変換手法の一つであるストレンクスリタクション方式を実施せずに請求項1記載の方法において、該ループを展開し、その結果できる展開後のループに対して新たにストレンクスリタクションを適用するループ最適化方法。

【請求項6】 請求項2に記載の条件で展開したループに対して、請求項4に記載のストレンクスリタクション方式を実施することを特徴とするループ最適化方法。

【請求項7】 請求項1に記載のループ展開方法において、該展開処理を実施するか否かを、展開によって生じるオブジェクトコードのサイズの増加量と、ユーザが指定するコンパイル時のオプションとの組合せによって決定することを特徴とするループ最適化方法。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明はコンピュータプログラムのコンパイラにおける最適化方式に係り、最適化コンパイラか、より効率的に実行されるようなコードを生成することを可能にする方法に関する。本発明はRISC

(Reduced Instruction Set Computer) に特に適応性を有しているが、一般のコンパイラにおいても利用可能である。

【0002】

【従来の技術】 コンパイラが生成するオブジェクトコードの実行時の効率を向上させる最適化技術が従来より種々提案されている。中でも、プログラム中で実行頻度が高く、実行時間の多くの割合を占める部分はループであるので、ループの最適化手法が特に重要である。これらの従来の最適化技法については、佐々正孝著「プログラミング言語処理系」(岩波講座ソフトウェア科学5、1989年)第9章441頁から499頁などに記載されている。

【0003】 このループ最適化手法の中で本発明に関するものとして、ループ本体を何回か複写して繰返しの終了判定の回数を削減するループ展開方法とループの繰返し本体内で実行される演算の削除/複雑度の軽減(より実行速度の速い演算による置き換え)操作を取り上げる。

【0004】 まず、ループの展開は、ループの繰返し数分だけ、もしくはその約数分だけその本体を複写して順に配置しすることである。例えば、次の100回繰返しのループ

```
loop 100回
```

```
ループ本体
```

```
end
```

を本体を4回複写して並べ、25回繰返しのループ

```
loop 25回
```

```
ループ本体
```

```
ループ本体
```

```
ループ本体
```

```
ループ本体
```

```
end
```

のようにすることがループ展開の典型である。ループ展開すると、上記の例では展開前に100回であった終了の判定を展開後には25回に削減できるので、実行速度を向上することかてきる。従来、この展開操作はループの繰返し回数かコンパイル時に既知である場合にのみ利用可能であった。

【0005】 次に、演算の強度の軽減についてである。これはループの繰返しにともなう定数値ずつ更新されるような変数(これを帰納変数と呼ぶ)の一次式に対し、その定数倍の部分において実施される乗算を加算に置き換えることを主とするものである。例えば、次のループ

```
loop
```

```
x = a[i]
```

```
i = i + 1
```

```
end
```

において、a[i]のアドレス計算では、通常、i*

「配列要素のサイズ(Lとする)」が生成される。ここで、iが帰納変数である。一般に乗算の計算コストは加算などに比べて大きいのでループ内でi*Lを計算した後i+1を行うかわりに、i-Lだけを行って繰返しを進行するものである。iが0から始まるものとする、次のように新たな変数i'を導入してこのループを変換する。

```

【0006】
    i' = 0 (iの初期値)
loop
    x = [] (&a+i')
    i' = i' + L
end

```

ここで、[]は添字操作を行う演算子で、ここでは引数(&a+i')で示される値をアドレスとして、配列要素の参照を実現するものである。また、&はその引数(ここではa)のアドレスを得る操作であるものとする。実際の計算機の命令では、&a+i'は&aを(ベースレジスタ、オフセット)で表現し、i'をインデックスレジスタで表現することになるが、RISC計算機ではベース、オフセット、インデックスという上記3種からなるアドレッシングの命令をもたないことが多い。そこで、さらに次のようにして、変数i''を導入してこれをベースレジスタの内容+オフセットに設定し、

```

    i'' = &a+0
loop
    x = [] (i'')
    i'' = i''+L
end

```

の形式で変換する。

【0007】これによりループ本体でコスト大の乗算か、よりコスト小の加算に置換され、実行速度が向上する。乗算に限らず、ループ内の加算などとは、それを削除することでもて、このような演算の強度(コストの大きさ)を軽減する変換操作はストレンクスリダクションと呼ばれる。

【0008】

【発明が解決しようとする課題】上記従来技術では、コンパイル時に実行回数が不明であるループは展開の対象としていなかった。さらに、ストレンクスリダクションにおいてはこれを実施するか実施しないかの二者択一でありループ展開と組み合わせて、展開後のある部分には適用し、ある部分には適用しないというように、これらを組み合わせる最適化方式は適用されていなかった。

【0009】さらに、ループ展開処理は一般にオブジェクトコードのサイズを増大させるので、ループ本体のサイズが一定値以下の場合にしか適用されず、実行速度とオブジェクトコードサイズの重要度に応じて柔軟に制御することができなかった。

【0010】本発明の目的は、コンパイル時には実行回

数不明のループにも展開操作を実施して実行速度を向上させ、この展開されたループに対してストレンクスリダクションを実施し、ループ構成命令を使用する機会を増やして一層の速度向上を得るとともに、実行速度とコードサイズとの重要度に応じて展開可否の条件をユーザが指定できる高性能なループ処理方式を提供するものである。

【0011】

【課題を解決するための手段】上記の目的を達成するために、コンパイラはコンパイル時に次の手順でループの展開、およびストレンクスリダクションを実施する。

(0)ループ本体のソース言語または中間語でのサイズとコンパイルオプションでの指定から、ループを展開すべきであるか否か、展開する場合の最大展開数を決定する。

(1)繰返し回数不明のループに対しては、回数がループ内で変更を受けないことを確認した後、そのループの入口で回数を判定し、その値によってそれぞれ対応する展開部分へ分岐するコード(「判定コード」と呼ぶ)を出力する。

【0012】(2)(1)での分岐のターゲットとなるよう、展開において複写したループ本体は、1回展開部、2回展開部、3回展開部、...のように、展開回数に応じて必要なだけ置く。(2-1)繰返し回数不明のループと展開数かループ繰返し数未満となるループでは、1回展開部、2回展開部、3回展開部、...とすべての展開部を置き、各展開部の最後にそれぞれ(1)の「判定コード」を置く。(2-2)繰返し回数かコンパイル時に既知で、かつ展開数か繰返し数以上の場合、繰返し数に等しい展開数の展開部だけを置く。例として、2回展開に限定する場合、(2-1)の条件に対応したものでは、1回展開部ではその最後で1回分の終了判定を置き、2回展開部ではその最後で終了判定してこの2回展開部自身を繰り返す形式となる。

(3)展開後のループに対して2回展開部では帰納変数+1、3回展開部では帰納変数+1、帰納変数+2、...のそれぞれを対象としてストレンクスリダクションを実施する。

【0013】

【作用】コンパイル時にループ繰返し回数か決定しているとき、実行時には、その繰返し回数もしくは最大の回数分に対応する展開後本体に分岐する。また、ループ入口の回数判定部分で実行時にループの繰返し回数が確定するときは、その回数を実行時にループ入口で判定して、対応する回数分の展開後本体に分岐する。展開後の本体では、上記最適化のための手段によってループの繰返し回数もしくはその因数分の実行文が配置されているので、終了判定無し、もしくは因数の個数分の判定回数で繰返しの実行を終了してループを脱出する。ここで繰返し回数の因数とは、それらの総和が繰返し回数に等し

くなるような整数という意味で使用している。

【0014】展開後の本体内では、帰納変数+1、帰納変数+2、...に対して、それらを添字とする配列参照で、配列の起点+それらをそれぞれベースアドレスとしてレジスタに保持しておき、ストレンジスリダクションを実施し、ループ本体内での演算を軽減する。

【0015】

【実施例】以下、図を用いて本発明の実施例を詳細に説明する。図1は本発明の一実施例を示すものである。コンパイラは図1にあるように、ソースコードの字句解析・構文解析(101)、意味解析・中間コード生成(102)、大域的最適化(103)と進行する。ここまでの処理およびコード生成(104)での処理は従来からのものである。

【0016】本発明は、大域的最適化(103)に改良を加えることであり、まずコンパイル時に繰返し回数が不明であるループを2回展開することを例にとり、図2のブロックラム片を用いて説明する。図2のループは繰返しを制御する変数*i*が0から*N*まで1ずつの刻みで動き、*i*を添字として配列*a*の要素を順次参照するものである。*N*の値はコンパイル時には既知でないものとする。コンパイル時の処理は以下のとおりである。

【0017】図1において、105のループ認識部で認識されたループ(図2 201)に対し、実行回数確定処理(106)を実施する。それに先だって展開条件検査部(107)において、このループを展開するべきかどうかを決定する。これは、今対象としているループの本体のサイズ(大域的最適化が処理対象とするブロックラムの中間表現でのサイズ)か、あらかじめ設定されている値を超えない時に展開可能とするものである。この設定値はユーザによって変更することかでき、コンパイルオプションによってより大きなもの、または制限なし等に変更できる。

【0018】以降の例では、ループが展開可能であるものとする。107では、対象としているループの実行回数かその入口点で決定され、ループ本体の実行時には変更を受けないことを検査する。ある種のフロッキング言語では言語仕様上で上記の条件が保証されているか、C言語などではそれが保証されておらず、コンパイラによる検査が必要である。この検査の手法は従来より実施されている最適化処理でのデータフロー解析を利用して行う。図2のループでは、繰返しの上限値を表現している変数*N*がループの本体内で設定されないことを確認することとなされる。

【0019】繰返し回数が確定すると、本ループが展開対象であるとされ、図1のループ展開部(108)で展開処理される。図3にその展開処理の詳細を示し、図4に展開結果のコードを示す。図3の処理を行うための確認処理の手順は以下の通りである：

「1」帰納変数の検出

ループ中で $i = i + C1$ ($C1$ は定数) という形式で一度だけ定義されている変数(これを帰納変数と呼ぶ)を見つける。

「2」帰納変数の初期値の検出

「1」で見つけた帰納変数のループ入口での値を求める。これを帰納変数の初期値と呼び、以下では10で表す。

「3」ループ終了判定用の比較式の識別

ループを終了する際の判定用の比較式の形式が「1」で見つけた帰納変数*i*を用いて $i < C2$ または $i \leq C2$ (ここで*C2*は定数またはループ中で値の変わらない変数) となっていることを確認する。

【0020】「4」繰返し回数の認識

10、*C1*、*C2*の組合せによって、繰返し回数がコンパイル時に既知となるか否か、および既知だとしたら偶数回か奇数回かが以下に示すように分かる。

場合1：<10定数、*C2*定数>

繰返し回数はコンパイル時に既知であり、 $(C2 - 10) / C1$ (判定用の比較式の種別が<の場合)あるいは $(C2 - 10 + 1) / C1$ (判定用の比較式の種別が<=の場合)である。

場合2：<10定数、*C2*定数でない>

場合3：<10定数でない、*C2*定数>

場合4：<10定数でない、*C2*定数でない>

これらの場合、繰返し回数はコンパイル時に既知でない。

【0021】「5」展開処理の実施(図3における処理)

回数判定コード生成部(301)および分岐コード生成部(302)において、上記手順によって得られた繰返し回数にしたがいそれぞれ対応する展開本体へ分岐するコード(図4 401、402)を生成する。さらに、ループ本体のサイズとコンパイル時のオプションから展開の回数を決定する。本例では2回の展開を実施し、繰返し回数の偶数/奇数を判定して分岐するものとする。

【0022】次に303~306によって本体の展開部分を配置する。本例では、ループ本体部分は1回展開部(本体そのまま(図4 403))と2回展開部(図4 406)とから構成される。展開された本体部では、展開数だけ帰納変数を更新する命令を配置する。306での処理により1回展開部の最後は、本ループが1回だけ繰返して終了する場合のための終了判定(図4 404、405)である。2回展開部の最後ではこの2回展開部自身を繰返すかどうかの判定コード(図4 407、408)を配置する。304、305、306の後処理ではこのように各展開での終了判定や新たな繰返しへの判定コードを生成する。なお、この配置は一例であり、奇数回のときに2回展開部の中間地点に飛び込むような配置も考えられる。

50 【0023】ここまでの、本例で出力されたオブジェク

トコードを実行した場合の実行過程を図4によって追うと、ループの入口において、

(a) 繰返し回数が奇数回のとき

401、402の判定、分岐によって、奇数の場合次に配置された1回展開部(403)を実行する。繰返しが1回のときは1回展開部の最後の判定(404、405)でループを脱出しラベル L@exit(409)に分岐する。3回以上のときはこの判定(404、405)で分岐せずに2回展開部を実行する。2回展開部では帰納変数 i を添字とした配列参照を i と i + 1 の二通りで行い、i 自身は2だけ更新するようにする。これに
10 関しては以下のストレンクスリタクションの節で述べる。この i の処理について、i での参照、1 だけ更新を2回繰り返す方式もある。2回展開部(406)の最後の判定部(407、408)で帰納変数が繰返しの上限値を超えたか否かの判定を実施してループを脱出する/繰返すの制御を行う。ループを繰り返す場合はこの2回展開部自身の先頭へ分岐するようラベル L@even を参照する。

【0024】(b) 繰返し回数が偶数のとき

401、402によって2回展開部に分岐し、以下
(a) の3回以上の場合と同様の動作を行う。このループでの具体的な命令語系列は図5の通りとなり、500がベースアドレスやカウンタの設定である前処理部で、401-501、402-502、403-503、404-504、405-505、406-506、407-507、408-508、のように図4と図5とが対応することになる。

【0025】次に、上記の例のコンパイル処理で展開されたループに対して、さらに最適化を行うために、コン
30 パイル時の処理として図1-109でストレンクスリタクション処理を実施することについて図5を用いて説明する。ここで、配列は添字値0から始まり、配列要素のサイズは4であるものとする。また、対象となる計算機の命令語にはインデクスモードによるアドレッシングは、「ベースレジスタの内容+インデクスレジスタの内容」を実効アドレスとするものだけしかないとする。

【0026】2回展開を実施すると、ソースコードのイメージでは、図2のフロケラムにおいては以下のように変換され

```
loop i = 0..N
  x = a[i]
  i = i+1
  x' = a[i]
  i = i+1
end
```

となり、a[i]の参照と i を-1更新した後の a
「i」の参照が生成される。x' は展開のために生じた x の複写である。ここで後者の a[i] の参照は、i を
-1更新しないで a[i-1] の参照とすることから

き、次のように a[i+1] が2回展開部に生成されることになる。

【0027】

```
loop i = 0..N
  x = a[i]      ----- (1)
  x' = a[i+1]   ----- (2)
  i = i+2
end
```

このうち前者の a[i] の参照(1)は i をインデクスレジスタに保持して、r1 をベースとするインデクスモードでの参照(506の最初の Loadx)で行う。a[i+1] の参照(2)に関しては、これを(a+1)[i]と考
20 えて、r2 をベースとし i をインデクスとするインデクスモードでの参照(506の2番目の Loadx)で行う。ここで(a+1)は配列 a の先頭から1要素分離れたアドレスを示すものとして考える。したがって、配列の起点アドレスとして a[0] および a[1] のアドレスをベースとしてレジスタ r1、r2 に保持するようにする(それぞれ500の2つの Loada)。500の2番目の Loada では配列要素のサイズが4であるので、r1 から4隔たった値として r2 を設定するようにしている。また、Loadc はインデクスレジスタの初期値設定、Shft1 は終了値を検査するために繰返し回数(rN)の4倍を設定するものである。これらの操作は前処理として当該ループの入口で繰返し前に実施する。

【0028】以上の処理により、506において、2つめの Loadx が元の a[i] の参照に対応するものの
30 ままであれば、i = i+1 がその直前にあるべきであるか、ここで a[i+1] に対応するものとしたので、この i = i+1 を削除することかてきる。ここでの操作は、元々は2つの Loadx に対する2つの i-1 が生成されるところを、そのうちの1つを省略して1つの i+2 にし、2に対して配列要素のサイズ4による乗算を+8の更新(506での Addc)としたストレンクスリタクションである。

【0029】図5が最終的に変換されたループの命令系列である。ここで、SP はベースレジスタであり、各命令はそれぞれ次のような意味をもつものとする。

40 Loada r, d(SP) … SP から d(定数)だけ隔たった地点のアドレス自身をレジスタ r にロードする
Loadc r, i … 定数 i をレジスタ r にロードする
Loadx r, x[b] … レジスタ b をベース、レジスタ x をインデクスとしてアドレスを得、そこに格納されている値をレジスタ r にロードする
Shft1 r, r, t … レジスタ r の値を左へ i ヒットシフトし、レジスタ t に格納する
50 Addc r, i … レジスタ r に定数 i を

加える

Test r …レジスタrの値を検査する

Comp c r, i …レジスタrの内容と定数iを比較する

Comp r, t …レジスタrの内容とレジスタtの内容を比較する

Bcond c, l …比較、検査結果がcの条件であるときラベルlに分岐する

次に、3回に展開する場合を例として考える。このとき 10
繰返し回数が定数で3回の場合、コンパイル時に301でそのことか判明するので、302においては3回分の展開部だけを生成する。繰返し回数に応じた分岐コードや終了判定コードは生成しない。

【0030】繰返し回数かコンパイル時には既知でない場合を、図6を用いて説明する。コンパイル時に図303の処理で1回、2回、3回の展開部を置く（それぞれ602、604、606）。302ではそのそれぞれに対応して、繰返し回数を判定して1回のとき、2回のとき、3回以上のときにそれぞれ対応する展開部へ分岐*20

B l …ラベルlに無条件分岐する

さらに、別の例として展開部を2のべき乗のものだけ用意して、繰返し回数の二進表現の各ビットが0であるか1であるかによって終了判定／分岐を行う例に関して図7を用いて述べる。はじめに説明した偶数／奇数でわけ 30
る2回展開の例は、これの特別な場合であるとも考えられる。ここで、繰返し回数はコンパイル時に既知でないものとする。

【0032】コンパイル時に展開部は本体自身（2の0乗回）と2回（2の1乗回）展開、4回（2の2乗回） 30
展開を用意する。303の処理で本体（702）、2回展開部（703）、4回展開部（704）を配置する。302では、繰返し回数の最下位ビットから順に「1」であるかを判定し、「1」であれば対応する展開部へ分岐するコード（701）を配置する。ただし、最終の展開部である4回展開部では繰返し回数が4以上を判定する（29ヒット目以降の上位ビットが「1」という判定でもよい）。

【0033】最終以外の各展開部の終了判定では、図306の処理として、自身に対応するヒット位置以下 40
は、実行時にここにきた段階ではすでに繰返し終了済となるので、コンパイル時の処理として、自身の次のヒット位置を判定する部分へ戻るようにする（702、703でのB命令）。最終である4回展開部の最後では4以上であるかの判定を行って自分自身を繰返すように命令を配置する（図305）。

【0034】

【発明の効果】本発明によれば、コンパイル時に繰返し

* 岐するコードを配置する。この分岐コードを図6601に示す。601は1と比較して等しければ分岐、2と比較して等しければ分岐、3と比較して以上であれば分岐という形式であるが、これ以外にインデックス分岐命令を使用する形式もありえる。最大展開数である3以外の各展開部の最後ではこれで繰返しが終了するので、ループを脱出するコード（603、605）を配置する。これは図3304、306での処理で実現されるものである。3回展開部（606）では、インデックスrcの更新に加え、繰返し回数rNを-3ずつ更新する。606の最後では、新たな繰返しのために、601の判定部を実行するようなコードを配置する。601に戻るよう分岐してもよいし、その判定部を新たに配置してもよい。ここでは新たに607を配置する形式とする。変換されたループの模式図は図6のようになる。

【0031】ここで、実行時に繰返し数が17回であったとすれば、5回展開部を3回、2回展開部を1回それぞれ実行してループを終了する。この例で導入した新たな命令は以下である。

回数の決定できないループも展開でき、ループ展開での最適化効果を得ることが可能になる。また展開されたループ本体に対して、帰納変数i、i+1、i+ 2、…を対象にストレンジスリダクションを実施することでiの加算を軽減することも可能になる。このストレンジスリダクションは多くのRISC計算機にみられるように、ベース+インデックス+オフセットでのアドレッシングモードをもたず、ベース+インデックスで対処すべき場合にも演算を減少させることかできるものである。

【図面の簡単な説明】

【図1】本発明の最適化操作を実施するコンパイラの構成図である。

【図2】本発明の適用例となるフロケラムである。

【図3】本発明の最適化操作を示す手順の図である。

【図4】本発明の一実施例によって展開されたループの構造を示す図である。

【図5】本発明を適用したコンパイラの出力コードの例である。

【図6】本発明を適用したコンパイラの出力コードの別の例である。

【図7】本発明を適用したコンパイラの出力コードの別の例である。

【符号の説明】

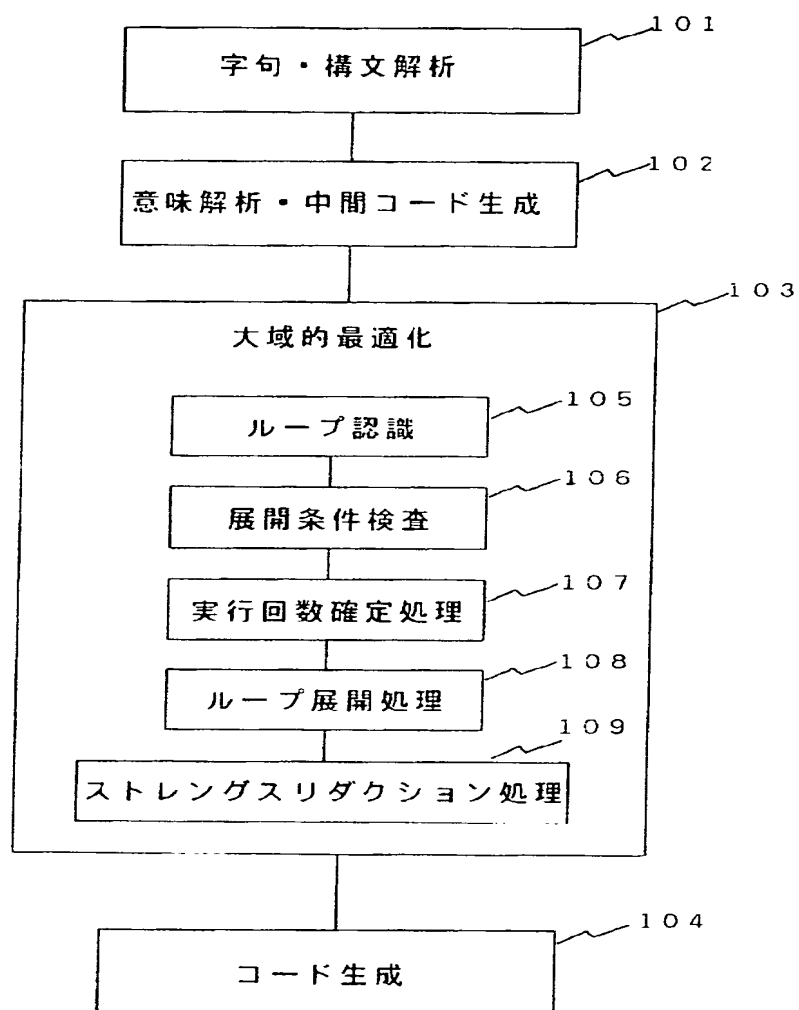
103…大域的最適化部、105…ループ認識部、106…実行回数確定処理部、107…ループ展開処理部、108…ストレンジスリダクション処理部。

【図1】

【図2】

図1

図2

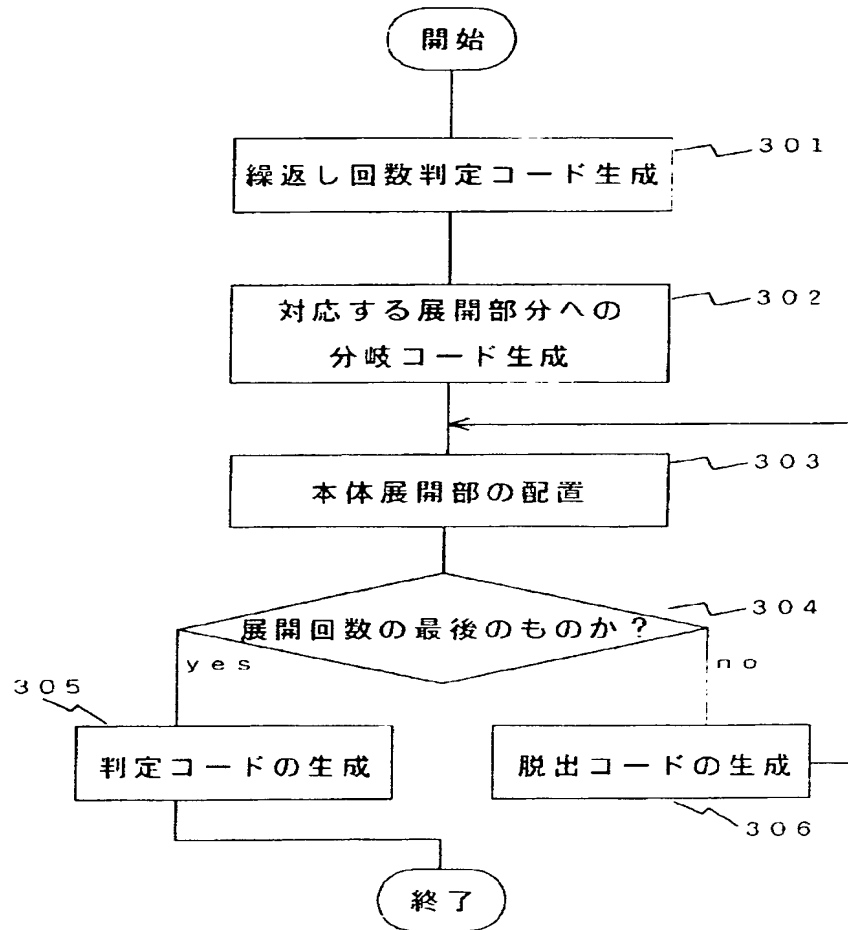


```
loop i = 0..N by 1
  x = a[i]
end
```

Figure 2 shows a code snippet for a loop, labeled 201.

【図3】

図3



【図4】

図 4



【図5】

図5

```

Loada r1,a(SP)      500
Loada r2,a+4(SP)
Loadc rc,0
Shftl rN,2,rN'

Test rN              501
Bcond LSB=0,L@even  502

Loadx rx,rc[r1]      503
Addc rc,1
Compc rc,1           504
Bcond =,L@exit       505

L@even:              506
Loadx rx,rc[r1]
Loadx rx',rc[r2]
Addc rc,8
Comp rc,rN'          507
Bcond <.,L@even      508

L@exit:              509

```

【図6】

図6

```

Loada r1,a(SP)      600
Loada r2,a+4(SP)
Loada r3,a+8(SP)
Loadc rc,0

L@0: Compc rN,1      601
     Bcond =,L@1
     Compc rN,2
     Bcond =,L@2
     Compc rN,3
     Bcond >=,L@3

L@1: Loadx rx,rc[r1] 602
     Addc rc,1
     B L@exit        603

L@2: Loadx rx,rc[r1] 604
     Loadx rx',rc[r2]
     Addc rc,4
     B L@exit        605

L@3: Loadx rx,rc[r1] 606
     Loadx rx',rc[r2]
     Loadx rx'',rc[r2]
     Addc rc,12
     Addc rN,-3

ラベルがない以外601に同じ 607

L@exit:

```

【図7】

図7

```

前処理              700

Test rN              701
Bcond bit31=1,L@1

L@4: Test rN
     Bcond bit30=1,L@2

L@5: Compc rN,4
     Bcond >=,L@3

L@1: ...              702
     B L@4

L@2: ...              703
     B L@5

L@3: ...              704
     Compc rN,4
     Bcond >=,L@3

```

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☒ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☒ GRAY SCALE DOCUMENTS
- ☒ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.